White Paper Report

Report ID: 106494

Application Number: HD-51568-12

Project Director: Travis Brown (trbrown@umd.edu)

Institution: University of Maryland, College Park

Reporting Period: 6/1/2012-5/31/2014

Report Due: 8/31/2014

Date Submitted: 9/26/2014

White Paper

Grant # HD-51568-12

ActiveOCR: Tightening the Loop in Human Computing for OCR Correction

Project Director: Travis Brown
University of Maryland

September 24, 2014

Introduction

The goal of the Active OCR project is to maximize the value of human input in the process of training an OCR (optical character recognition) system. Off-the-shelf OCR applications often perform poorly on facsimile images of historical documents for a number of reasons (including image quality), but the most important is that they typically ship with models that are trained on contemporary texts. This means that historical type faces, orthography, and vocabulary all pose significant challenges for these systems (or at least these models), and the rate of error can be high enough to render the use of OCR transcription useless for many forms of literary or historical analysis.

Many OCR systems allow users to develop their own training materials and create their own models, but this process is time-consuming and expensive. Over the past decade a large number of tools have been developed that aim to make this process easier—there are at least a dozen training material editors for Tesseract (an open source OCR system), for example. Most of these tools primarily support the creation of training material through “ground-truthing”—that is, taking a facsimile image and manually correcting its transcription in detail. This process often requires users to identify accurate bounding boxes at the word or even the character level.

The Active OCR project takes a different approach to developing training material. Instead of taking entire facsimile images of pages from a source document and correcting them from start to finish, it supports the creation of synthetic training pages that are made up of selected words (and in some cases other units of text) that the Active OCR system and its human contributors have agreed on.

We run several OCR engines on a single set of documents and then compare their output in order to identify locations (usually at the word level) where the systems disagree, and where human input can be most effective. These points for adjudication are presented to the human contributor in a simple web interface that is designed for speed and ease of use; in the current implementation it is possible to for an untrained participant to provide several hundred judgements in ten minutes with a reasonably high degree of accuracy.

Once a sufficient number of words are given trustworthy transcriptions, the system generates a set of synthetic training pages and saves them in the formats required for the underlying OCR engines. The new models trained by these materials can then be used to transcribe the original text again. The resulting transcriptions can be fed back into the system, potentially beginning another cycle of human-assisted correction.

OCR systems and formats

The two most widely used open source OCR applications are Tesseract, which is now developed by Google, and Ocropus. Earlier versions of Ocropus used Tesseract for character recognition, but for the last three minor versions it has relied on its own character recognition engine. This fact that these two applications are independent (and take different approaches) means that they are good candidates for components of a joint system: while they do occasionally make identical mistakes, they tend not to. Active OCR currently uses Tesseract 3.04 and Ocropus 0.7.

The focus of the Active OCR project is on improving OCR for eighteenth-century texts, and our core test set is a collection of eighteenth-century volumes from the HathiTrust Digital Library (many of which were digitized by the University of Michigan, and all which are in the public domain[1]). Some of these volumes include OCR transcriptions produced by ABBYY FineReader, a commercial, closed source OCR system that often outperforms Tesseract and Ocropus. While we cannot retrain FineReader, the high quality of these transcriptions (which have no commercial restrictions) relative to ones produced by Tesseract or Ocropus means that they are another useful source of information for the system.

All three systems provide output in the hOCR format, an open standard for OCR output that embeds transcriptions in HTML. They vary widely in their use of the capabilities of hOCR, however. FineReader (at least in 8.0.1, the version used to produce our transcriptions) provides word-level bounding boxes and some word-level style information (specifically font size, style, and family, although only the font style seems to be reliable). FineReader also divides the page into blocks and paragraphs, making it possible in many cases to distinguish page headers. Ocropus only provides line-level bounding boxes, and does not include any information about style or layout. Tesseract provides some layout information, although this is often inaccurate, and includes word-level bounding boxes and a word-level confidence score.

Unlike the current version of Ocropus, the Tesseract system also computes character-level bounding boxes, and while it does not include these boxes in its hOCR output, it can also generate “box files” that do capture this information. In the early stages of the Active OCR project we developed a fork of Tesseract that represented character-level bounding boxes in the hOCR output using the standard’s “character segmentation cut” vocabulary. We eventually moved away from this approach for a number of reasons, including the complexity and underspecification of the character cut vocabulary, the fact that this part of the hOCR standard is not used by any other OCR system (to our knowledge), and the challenge of maintaining a fork of a large project like Tesseract. In our current system we accomplish the same goal more simply (but less elegantly) by generating and parsing both the hOCR and box file output from Tesseract.

There are a number of other differences in the format of the output from these systems (they all use different coordinate systems, for example—Tesseract is inconsistent in this respect even between its two formats). While there are several libraries designed to parse hOCR, none of them capture all of the information we need across all three flavors we have to deal with, so we use a custom parser to transform our hOCR transcriptions into an internal representation that

[1] Note however that there are some restrictions on the redistribution of images from Google-digitized HathiTrust volumes in the public domain. The Active OCR application does not present any images to unauthenticated users, and it does not ever present the user with more than a few words in a single image.

smooths over the inconsistencies and allows us to compare transcriptions despite the differences in granularity and detail.

Tesseract and Ocropus also differ greatly in their training processes and their requirements for training data. Tesseract expects box files paired with images, at least roughly at the page level, while Ocropus requires line images paired with transcriptions in plain text files (i.e., Ocropus does not require any bounding box information for training). We provide tools for extracting images from the original corpus and composing them and pairing them with transcriptions as required by each of these systems.

There is some guesswork involved in this process; for example, we have found that Ocropus is very sensitive to word spacing in the training material, and for Tesseract we came to the conclusion after rough experimentation that synthetic training pages about half the height of our source pages and one quarter their width seem to work well. These numbers are currently parameters to the system, and are likely to require manual tuning when adapting the system to other domains, or even other facsimile sources.

Comparing OCR output

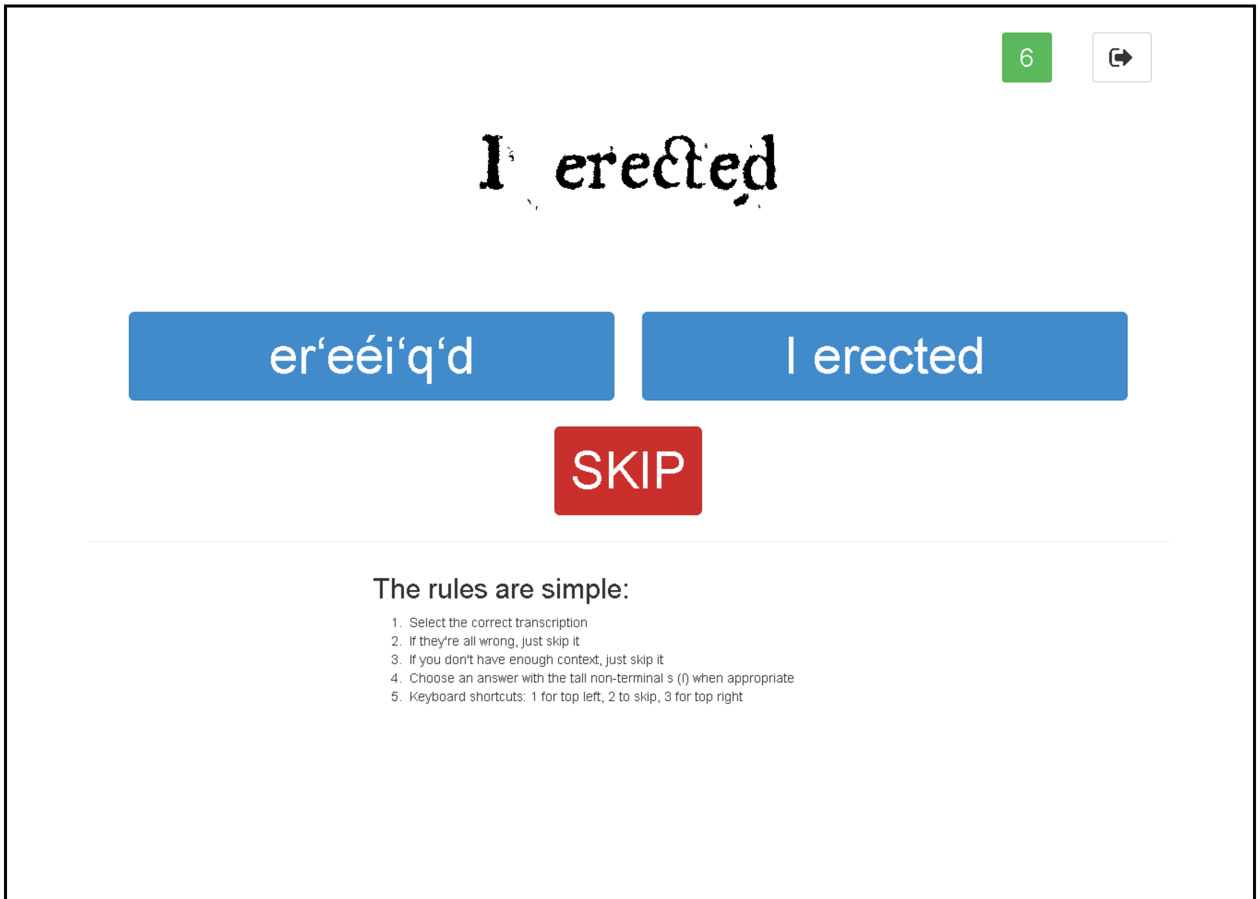
The difference in granularity between the output of our three systems posed a number of challenges when we wish to identify decisions we need a human to make. Because in the worst case we only have line-level bounding boxes, we have to begin alignment there. For every page in the corpus we identify triples of lines that share (pairwise) a configurable minimum proportion of their area. This approach means that we may throw out some word triples that are good matches (if for example one system splits a line but the others do not), but in practice we have found that this is relatively rare, and that the systems tend to agree at the line level.

Next we need to use the most finely-grained bounding box information from Tesseract to estimate character bounding boxes in the other two transcriptions for each line. We use our own implementation of Hirschberg’s sequence alignment algorithm to align the character sequences from Ocropus and FineReader with the sequence from Tesseract. For example, if Tesseract transcribes a line as “Without: farther compliment to the” and Ocropus as “without faither compliment to the”, the output of Hirschberg’s algorithm allows us to recover word and character-level bounding boxes for the final word “the” in the Ocropus transcription, despite the extra character in the Tesseract output and the errors in both.

Hirschberg’s algorithm takes as a parameter a cost function for the standard sequence manipulation operations (insertion, replacement, and deletion). We currently use a simple cost function that does not attempt to take knowledge of our domain into account. This has advantages in terms of portability, and quality of alignment does not seem to be a concern in our application, but a smarter cost function may produce better results.

The algorithm also computes a cost associated with each alignment, which gives us an intuitive measure of how closely the three systems agree on each line. We use this measure in our calculations of which words to feed to humans for review: we prioritize words with a high word-level total difference across engines and a low line-level difference.

We now have a way to map between locations in the three different transcriptions for each line. We use Tesseract’s judgements on words to create a list of word triples that we insert into our database, along with their associated word and line-level costs. Relying on one engine’s word boundaries results in some word-boundary errors in our database, of course. For example, in the figure below Tesseract has transcribed “I erected” as “er’eéi’q’d”, while both Ocropus and FineReader provide the correct transcription. We do not see any reason to be concerned about occasionally presenting users with judgements about (small) multi-word sequences, however, and these may actually improve the training output, since the resulting synthetic lines will contain some natural word spaces. Improving word identification would be a fairly straightforward modification to the system if desired.



The rules are simple:

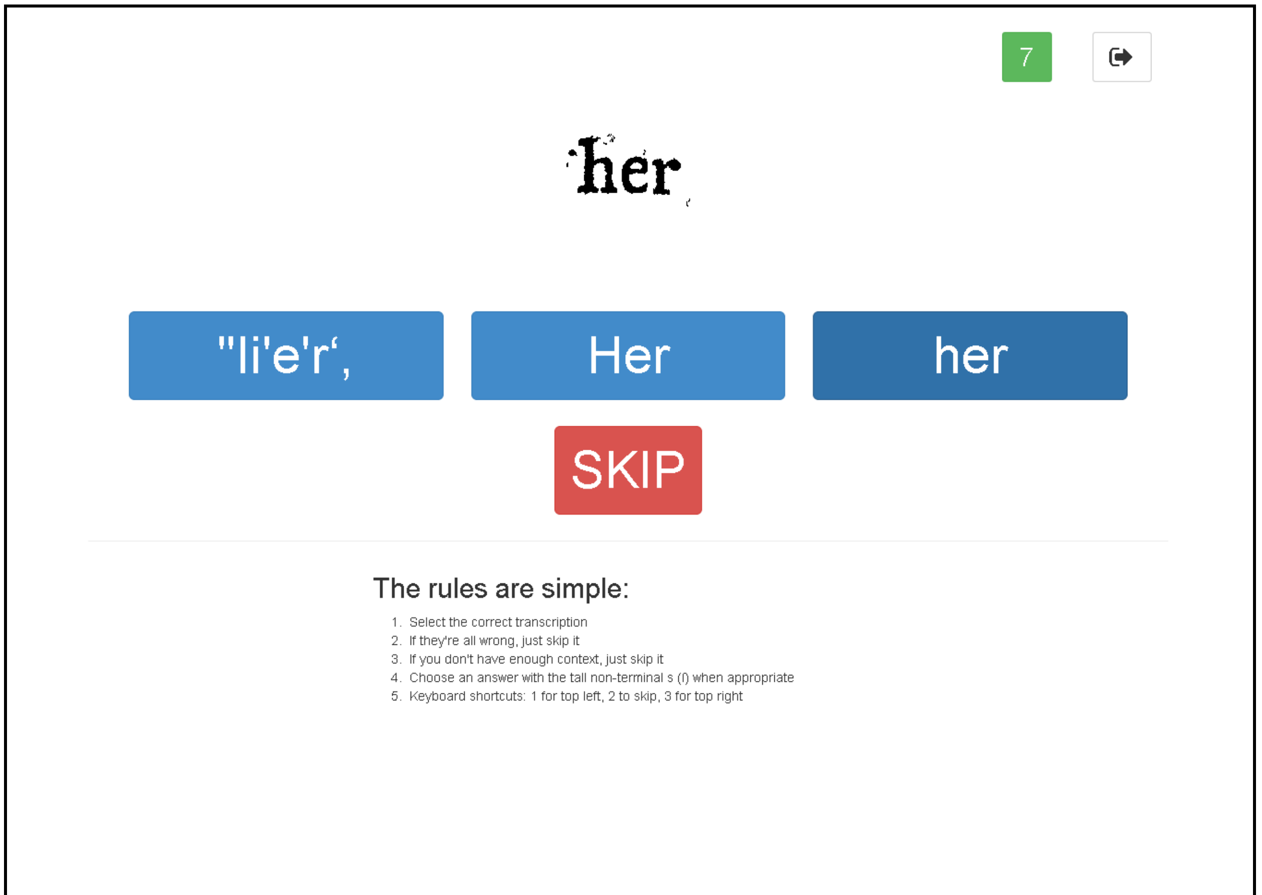
1. Select the correct transcription
2. If they're all wrong, just skip it
3. If you don't have enough context, just skip it
4. Choose an answer with the tall non-terminal s (ſ) when appropriate
5. Keyboard shortcuts: 1 for top left, 2 to skip, 3 for top right

Figure 1: Multi-word sequences in the interface due to word errors

The Active OCR interface

The primary goal of the Active OCR interface is to facilitate quick identification of correct transcriptions in a set of candidates proposed by the system. We use a number of factors to prioritize words: both line cost and word cost (as discussed above), whether the word has been seen by a human before, and whether at least two systems agree (or nearly agree) on a transcription. The job of the human is simply to process the stream of words—he or she should not have to worry the prioritization or any other unnecessary details.

The interface is accordingly very simple: the user is presented with a cropped image representing a single word (or small multi-word sequence), together with several buttons for candidate transcriptions and a large “skip” button. He or she is asked to select the correct transcription if one is provided, and if not (or if more context is necessary) to skip the word. We provide keyboard shortcuts to save effort, and the application is designed with ease of use on smartphones in mind.



The rules are simple:

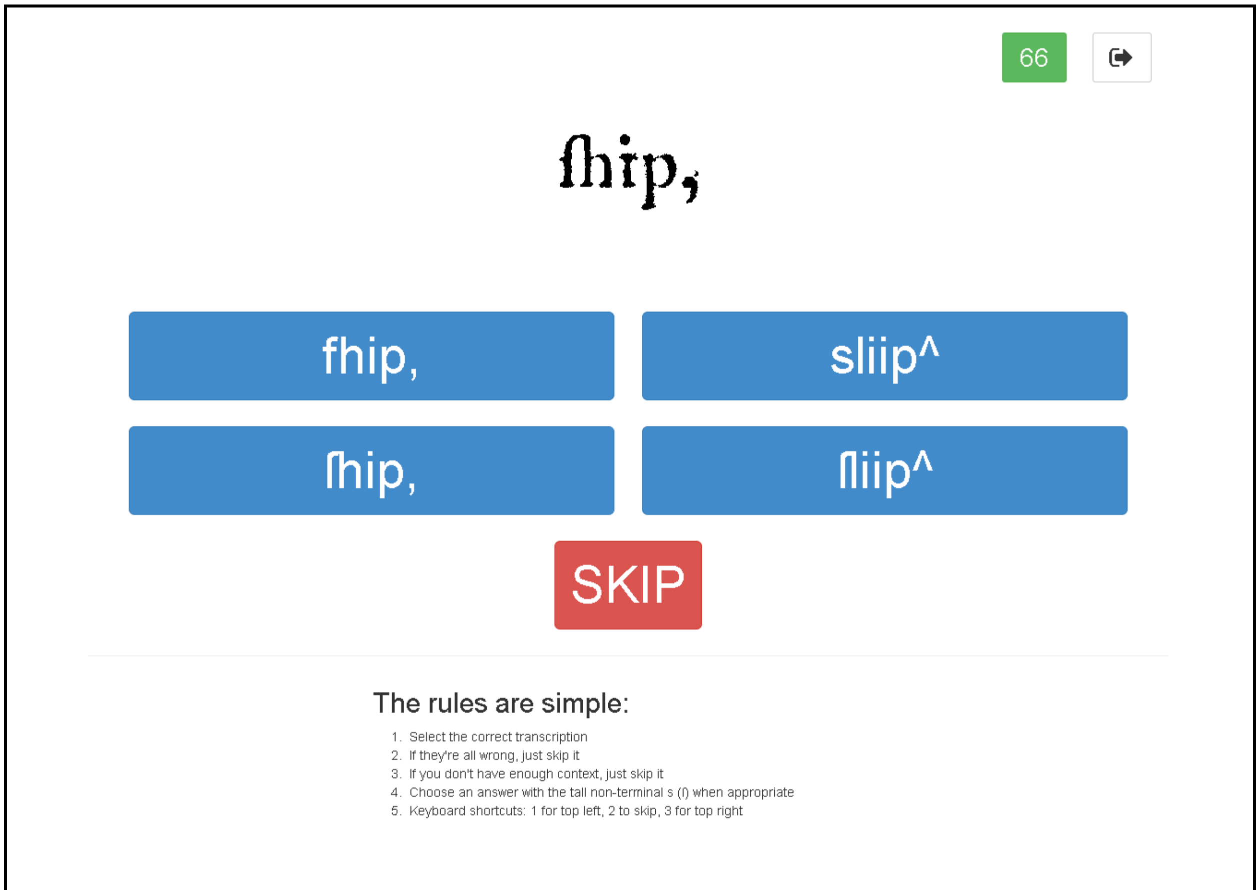
1. Select the correct transcription
2. If they're all wrong, just skip it
3. If you don't have enough context, just skip it
4. Choose an answer with the tall non-terminal s (ſ) when appropriate
5. Keyboard shortcuts: 1 for top left, 2 to skip, 3 for top right

Figure 2: Making a selection

In early prototypes we allowed the user to enter corrections manually in cases where none of the candidate transcriptions were correct. We found that this substantially slowed the speed of identification, however, and in many cases the resulting correction cannot be used for training because of the need for character-level bounding boxes. Adding a system that would allow users to input character bounding boxes would be even more at odds with our goals for the system, so we dropped manual correction entirely in the current version.

While some users report that liberally skipping words initially feels disconcerting, it quickly becomes natural, and is well-motivated: for even small installation of a single volume, there are dozens or hundreds of thousands of aligned words for review. Words that are skipped are marked as such and are less likely to be shown to future users.

We have also implemented a mechanism for providing additional candidates in cases where the domain expert has identified systematic errors across the OCR systems. For example, many eighteenth-century documents use the tall non-terminal *s* (ſ), which is typically recognized by OCR systems either as *f* or (more rarely) as *s*. If we wish to use the Unicode character in our training materials, we can add a component that adds candidates to the list presented to the user.



The rules are simple:

1. Select the correct transcription
2. If they're all wrong, just skip it
3. If you don't have enough context, just skip it
4. Choose an answer with the tall non-terminal s (ſ) when appropriate
5. Keyboard shortcuts: 1 for top left, 2 to skip, 3 for top right

Figure 3: Generating additional candidate transcriptions

In figure 3, for example, the word “ship” has been transcribed as “fhip” and “sliip^”, but the Active OCR system has also suggested additional candidate transcriptions with the tall non-terminal *s*, including the correct “ſhip”.

At any moment during human adjudication a site administration can generate new sets of training materials for Tesseract and Ocropus. Figure 4 shows an example of a synthetic line from a training set. This line does not make sense as a phrase of English, since it is a composite of words from a number of different lines, but that is not relevant to the training systems. Note that there are some minor discrepancies in the vertical placement of the words: the word “till” for example is higher off the baseline than it would be likely ever to be in naturally printed text. We have not found this to be a problem yet, but it could be corrected with some relatively simple analysis of the baseline of individual words.

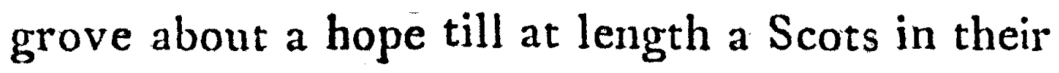

grove about a hope till at length a Scots in their

Figure 4: Synthetic training image output

The Active OCR software

The Active OCR web application and all associated tools are written in Java and Scala, and the web application is built on the Play Framework. Authentication is managed by the SecureSocial library, which allows users to log in using social identity providers through OAuth 1.0 and OAuth 2.0. The site currently only allows authentication via Twitter and GitHub, but SecureSocial provides integration for a wide range of additional identity providers, including Facebook, Google, and LinkedIn, and support for these could be added to Active OCR with some minor changes to configuration. The project code and documentation are available at GitHub and are released under the Apache License, Version 2.0.